Welcome To …

x64 Deep Dive

Microsoft Developer Support
Global Escalation Summit, 2010

Presented by :

T.Roy

CodeMachine Inc.

www.codemachine.com

Speaker Introduction

- T.Roy
  - Masters Degree in Computer Engineering
  - 20 years experience in system software development
  - 10 years international teaching experience
  - Specialization in Windows Driver Development and Debugging
  - Founder of CodeMachine
- CodeMachine Inc.
  - Consulting and Training Company
  - Based in Palo Alto, CA, USA
  - Custom Driver Development and Debugging Services
  - Corporate on-site training in Windows Internals, Networking, Device Drivers and Debugging
  - http://www.codemachine.com

CodeMachine Courses

- Internals Track
  - Windows User Mode Internals
  - Windows Kernel Mode Internals
- Debugging Track
  - Windows Basic Debugging
  - Windows User Mode Debugging
  - Windows Kernel Mode Debugging
- Development Track
  - Windows Network Drivers
  - Windows Kernel Software Drivers
  - Windows Kernel Filter Drivers
  - Windows Driver Model (WDM)
  - Windows Driver Framework (KMDF)

Agenda

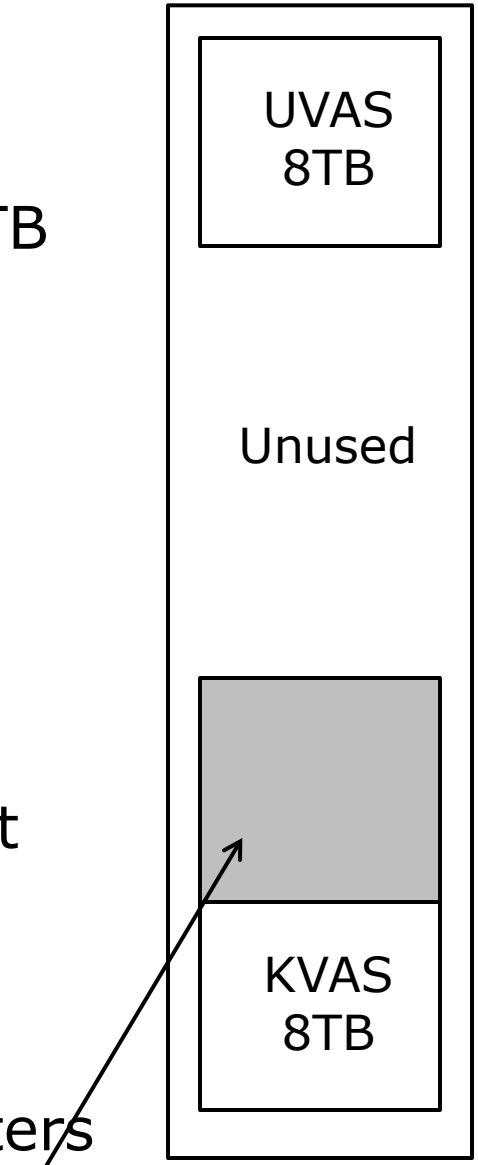
- x64 Architecture
- x64 Compiler
- x64 Call Stacks
- x64 Debugging

Register Changes

- Registers
  - Contain data and addresses
  - Determines the size of the data bus
  - Determines width of data processed by CPU instructions
    - Makes the x64 CPU a TRUE 64-bit CPU
- Non Volatile Registers
  - RBX, RBP, RSI, RDI, R12-R15
- Register Based Parameter Passing
  - RCX, RDX, R8, R9
- RBP not used as the frame pointer
- Segment Registers DS, ES, SS are not used
  - CS used for attributes only
  - GS register used to access TEB and PCR
- Debugger .trap command shows partial context

Virtual address space

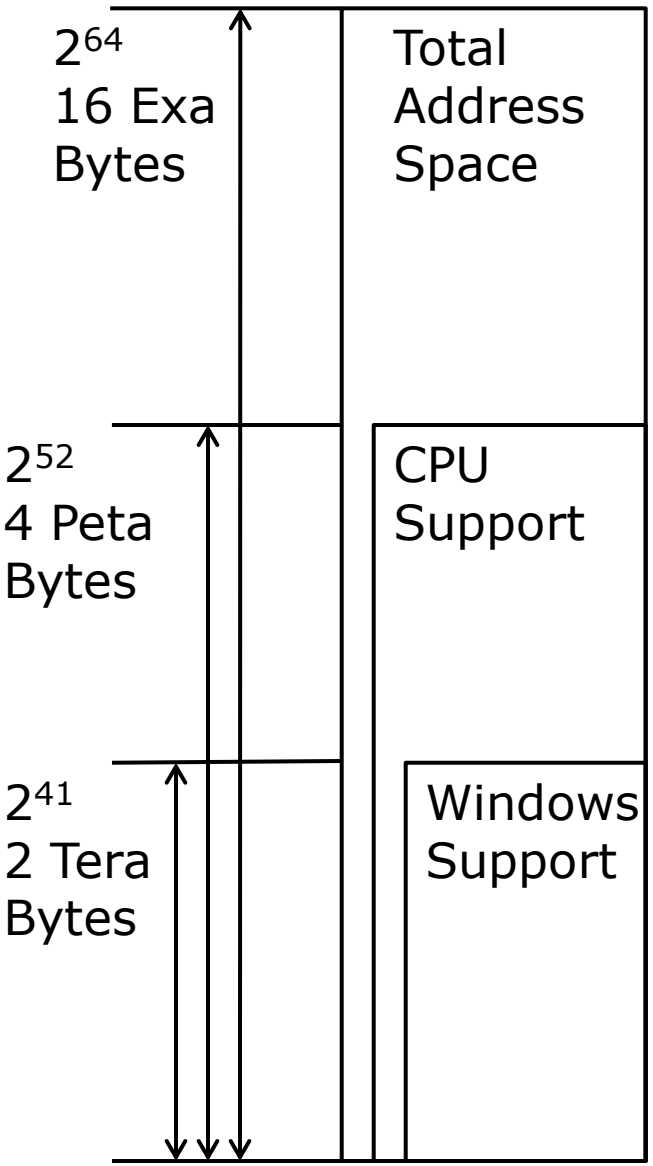
- X64 virtual address are 64 bits, but
  - CPU uses only the lower 48 bits
  - Which limits the address range to 256 TB
- Windows uses only 44 bits (mostly)
  - Restricts the user mode VAS to 8TB
  - Restricts the kernel mode VAS to 8TB
- But there is stuff kept outside this 8TB range in kernel virtual address space
  - Like Hyperspace , PTEs, WSLE etc.
  - Cannot contain data that require 128 bit atomic instruction to access
    - E.g. CMPXCHG16B
  - Required to manipulate push locks, interlocked SLISTS, EX_FAST_REF pointers



KVAS used internally by Windows 248TB

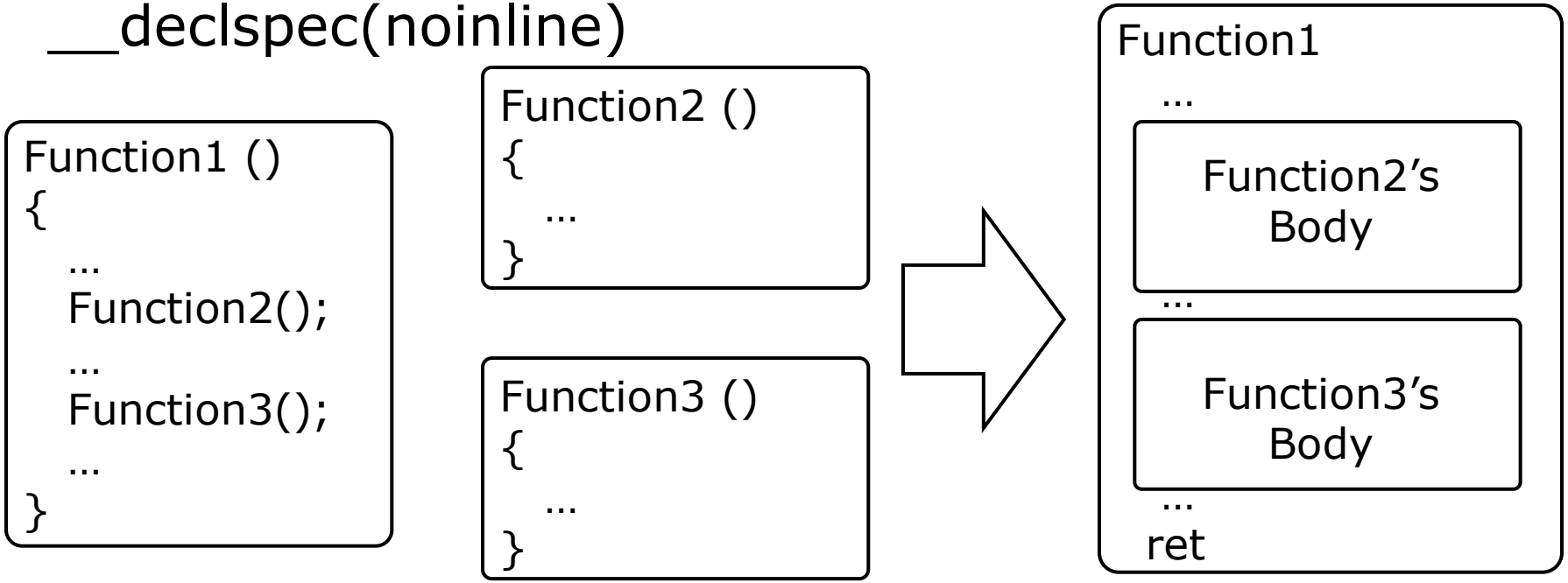
Physical address space

- x64 Physical Address are 64 bit, but
  - CPU decodes on 52 bits of the physical address
- PTEs contain a 36 bit PFN
  - Page size is 4K (12 bits)
  - Maximum addressable physical space is (48 bits) or 256TB
- Windows supports up to 2TB of physical RAM
  - Requires 41 bits of physical address



Function Inlining

- X64 compiler is very aggressive about inlining functions
- Avoids the overhead of function call and stack setup
  - But increases the size of the executable file
- Controlled by /Ob compiler flag
- Can be disabled on a per function basis by __declspec(noinline)



Tail Call Optimization

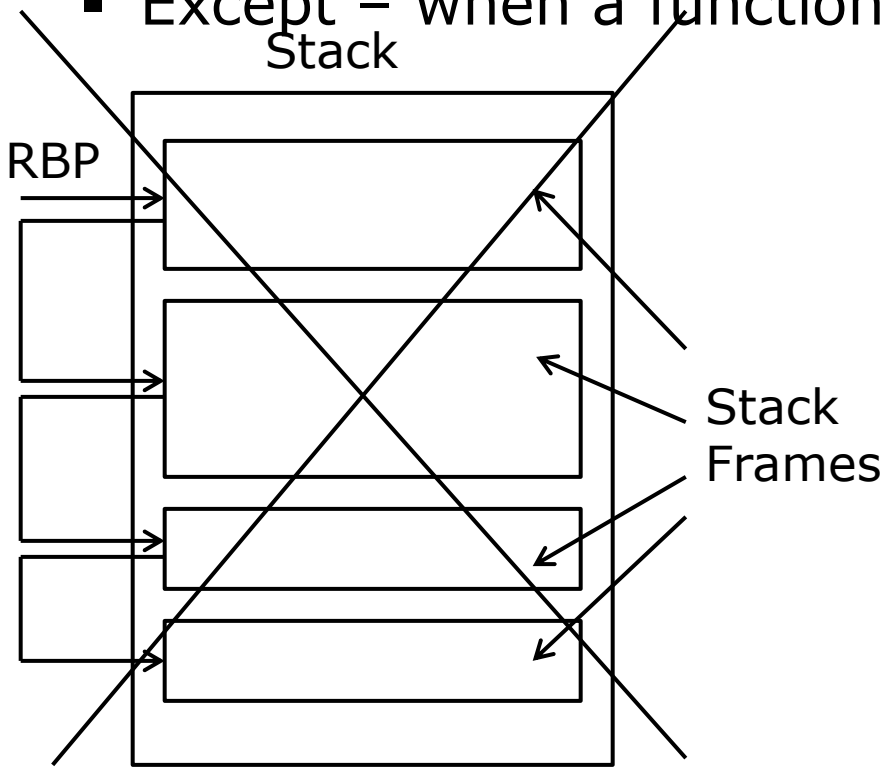
- X64 compiler can optimize the last call made from a function by replacing it with a jmp
- Avoids overhead of setting up stack frame for callee
- Caller and callee share the same stack frame
- Callee returns directly to the caller's caller

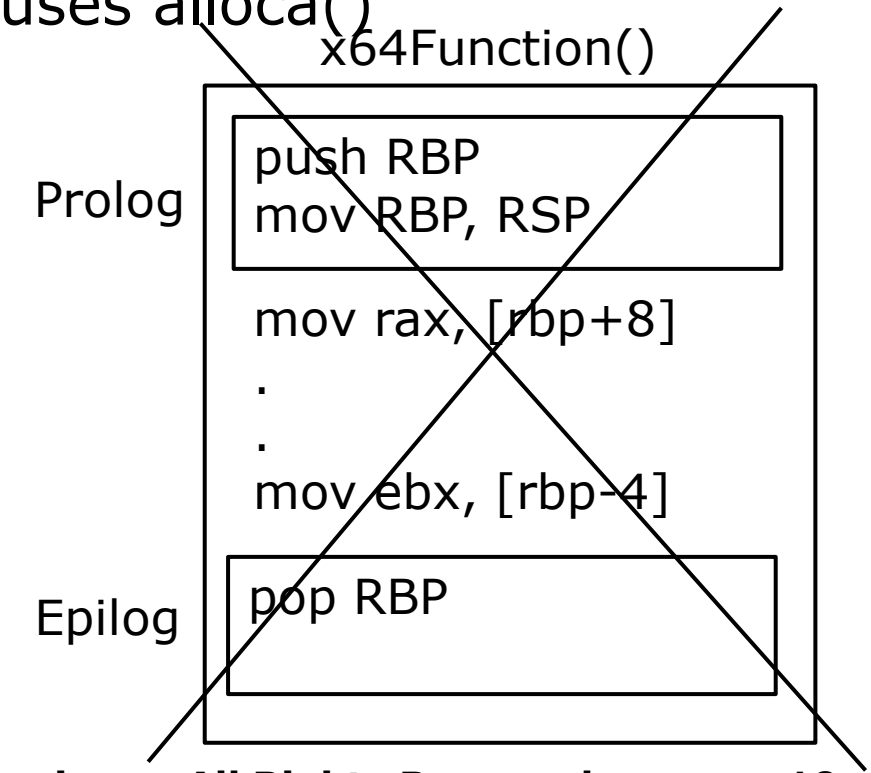
```
Function1 ( P1, P2)
{
   …
   Function2(P1);
   …
   Function3(P2);
   …
   return Function4(P1, P2);
}
```

```
Function1
   …
   call Function2
   …
   call Function3
   …
   jmp Function4
```

No Frame Pointer

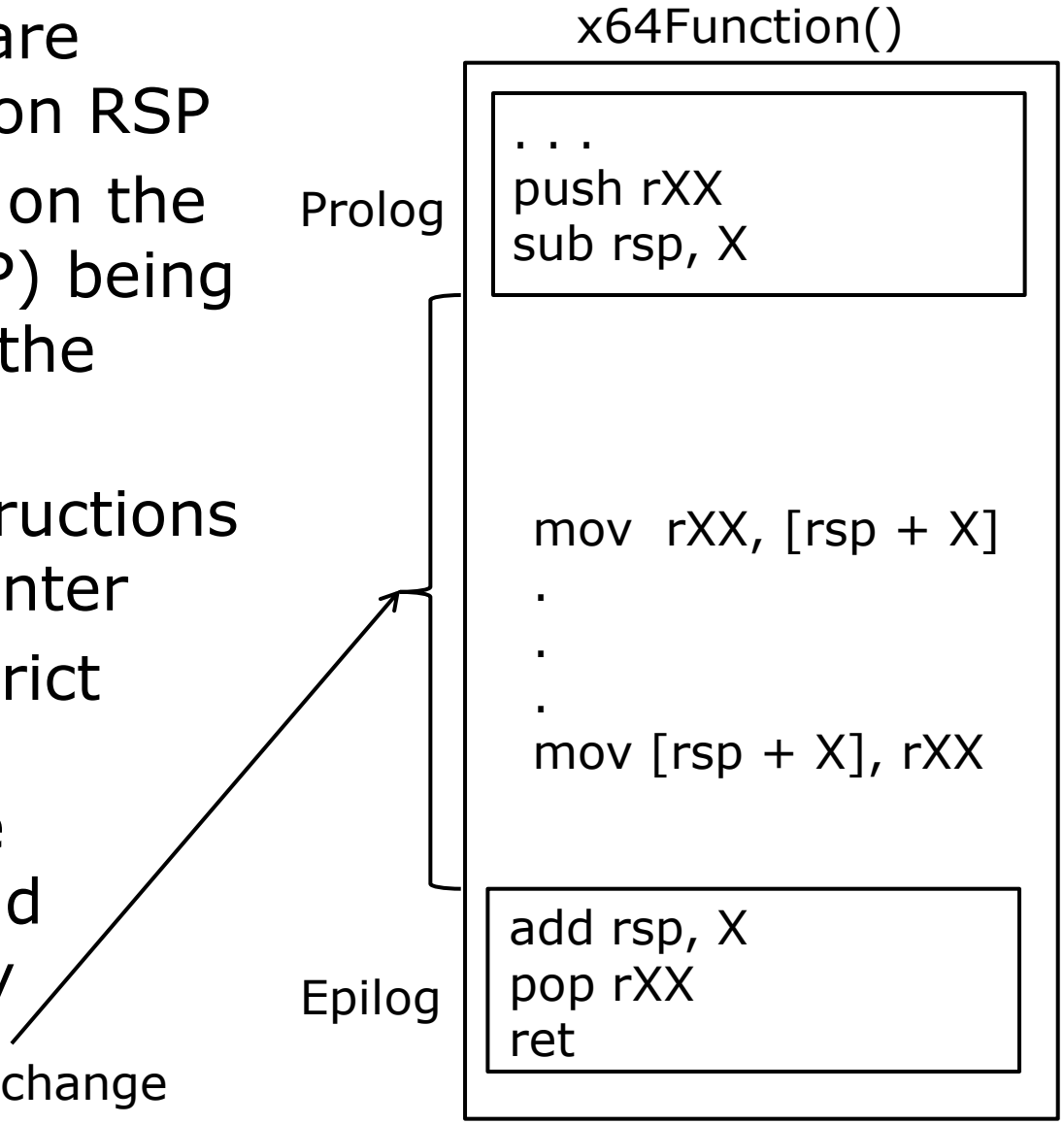
- X64 functions use the stack pointer (RSP) to access stack based parameters and local variables
- No need for a separate frame pointer
  - No FP and hence no FPO (Frame Pointer Optimization)
    - RBP is now a general purpose register
  - Except – when a function uses alloca()





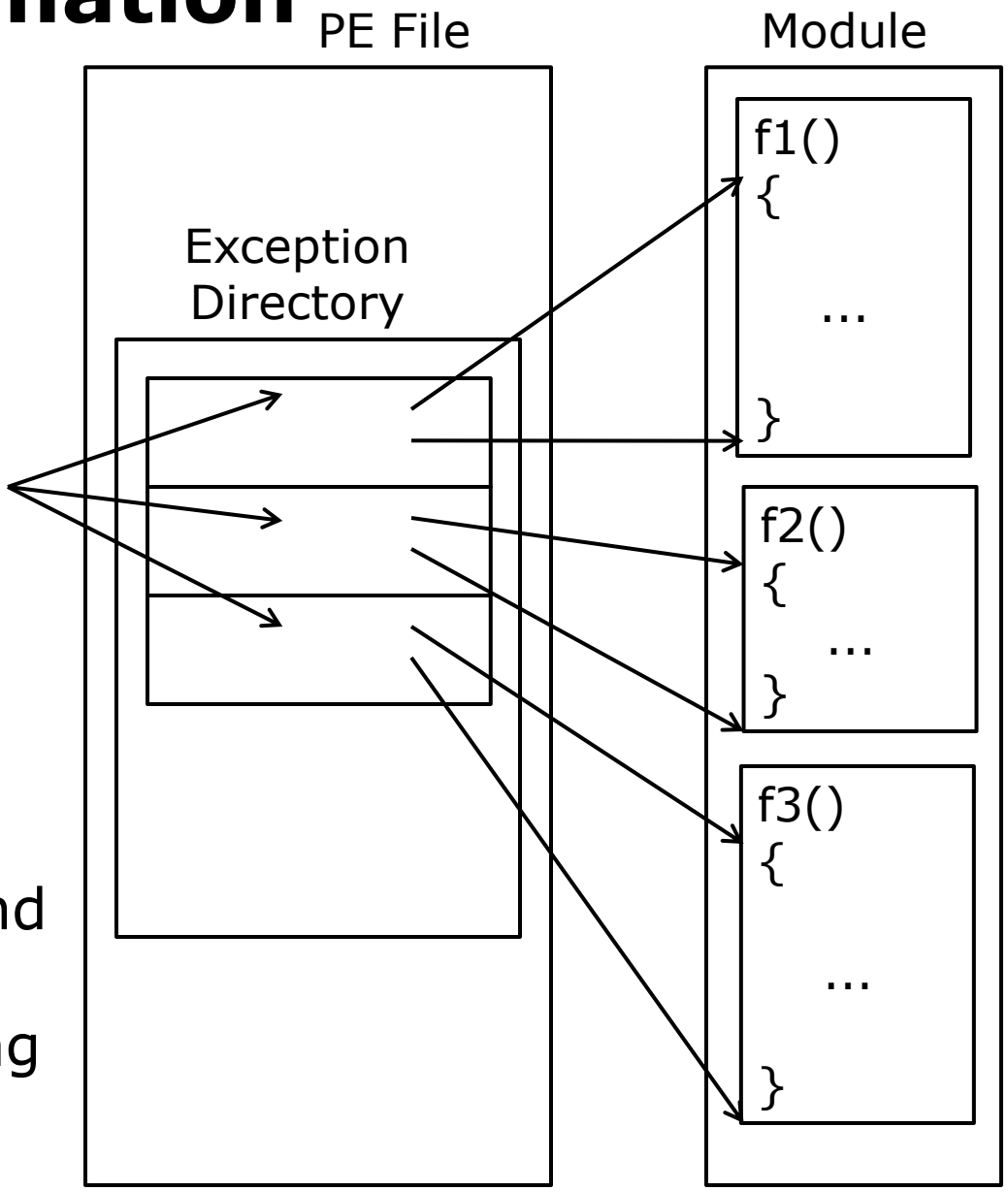
Static Stack Pointer

- Stack references are performed based on RSP
- Functions depend on the stack pointer (RSP) being static throughout the function body
- Push and Pop instructions alter the stack pointer
- x64 functions restrict push and pop instructions to the function prolog and epilog respectively



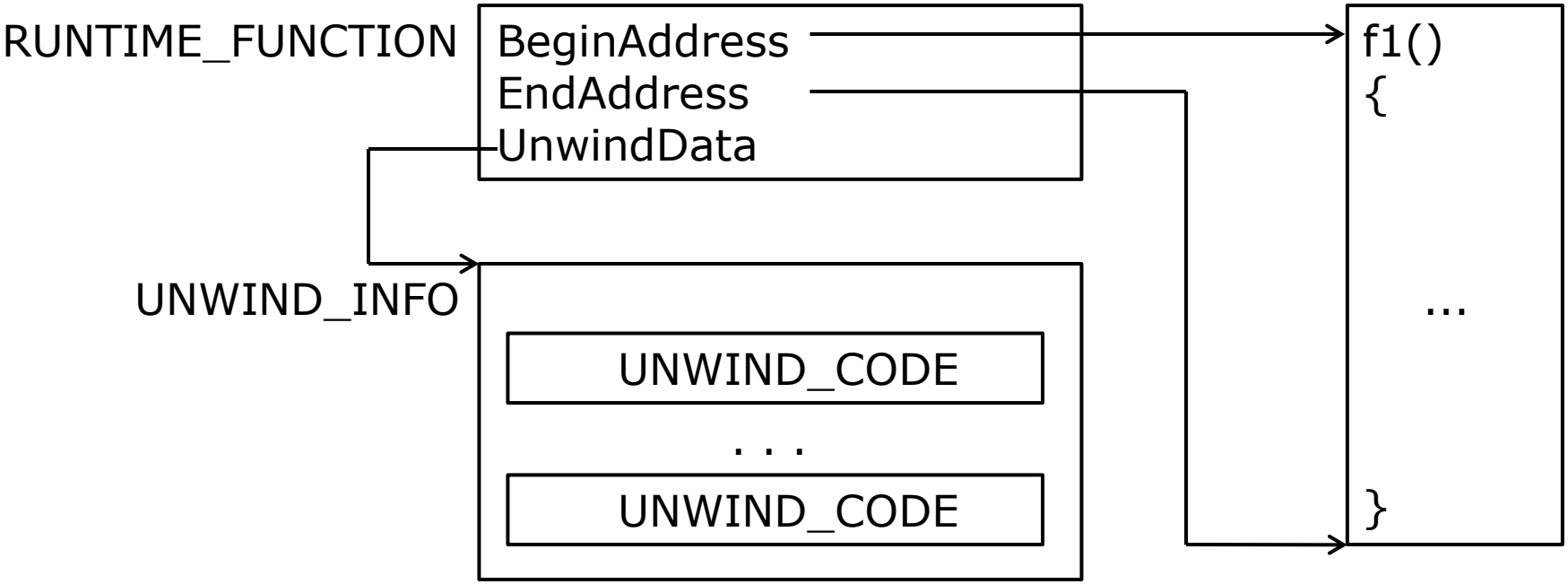
Exception Information

- X64 PE files (PE32+) contain Exception Directory (.pdata)
- Exception Directory contains RUNTIME_FUNCTION structure for every non-leaf function
- RUNTIME_FUNCTION
  - Function extents
  - Points to stack unwind information required for exception handling
  - Points to exception handler



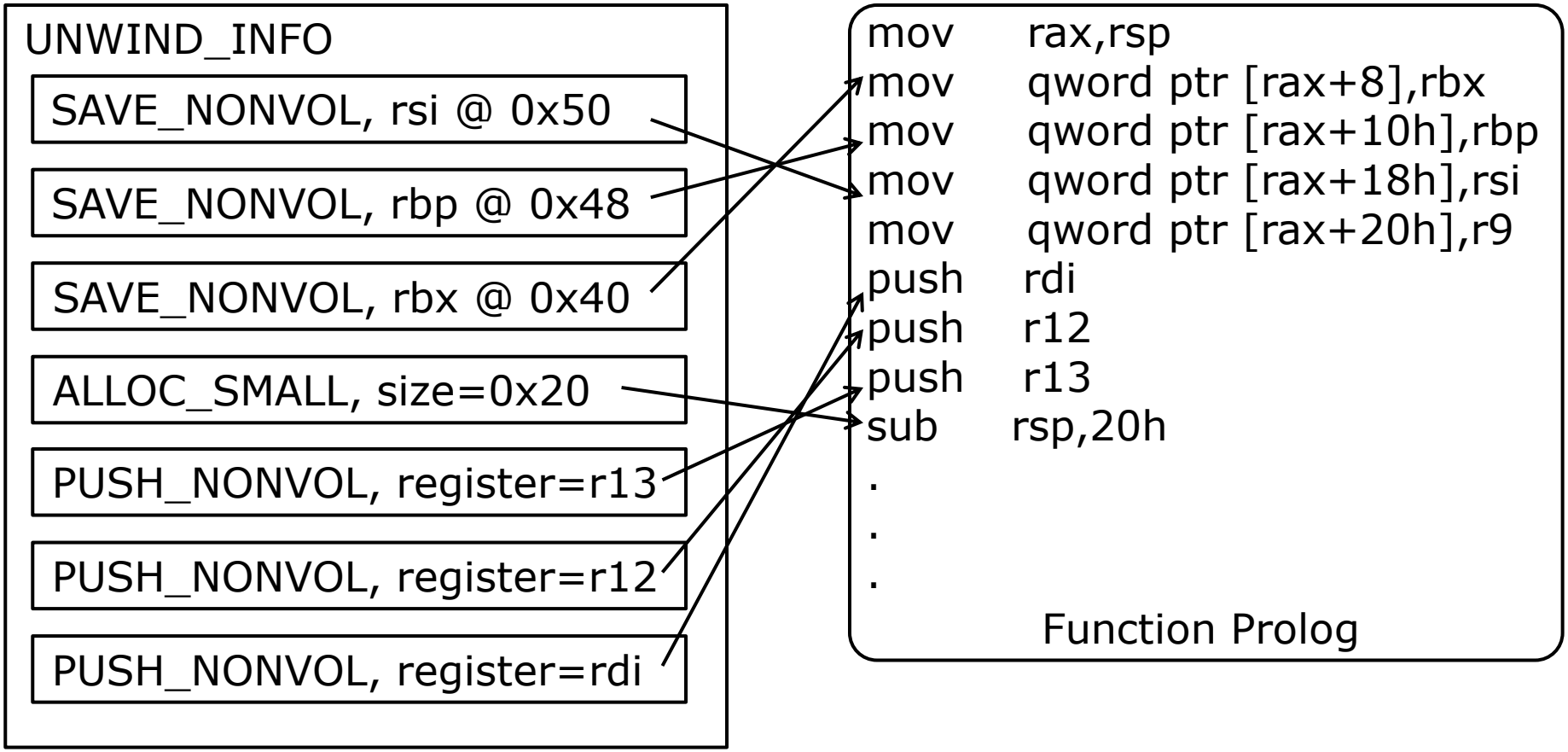
Stack Unwind Information

- UNWIND_INFO describes function call stack usage
- Identifies locations on the stack where function saves non-volatile registers, stores local variables
- Contains variable number of embedded UNWIND_CODE structures



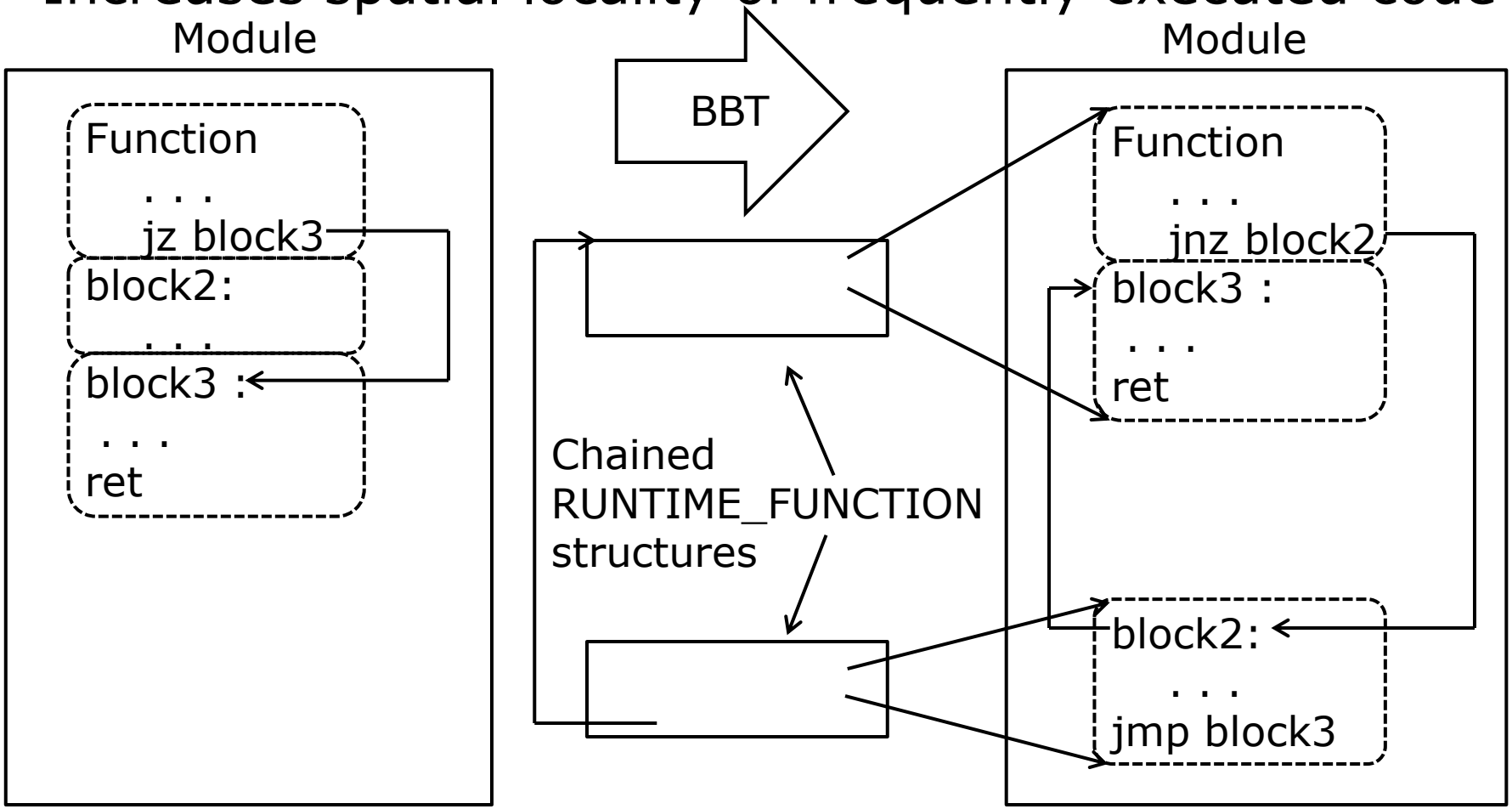
Unwind Code

- Each UNWIND_CODE structure describes one stack operation performed by the function's prolog
- Order of UNWIND_CODEs is important
  - In reverse order of operations performed by the prolog



Performance Optimization

- Post link phase profile guided optimization applied to OS binaries (aka BBT)
- Increases spatial locality of frequently executed code



Parameter Passing

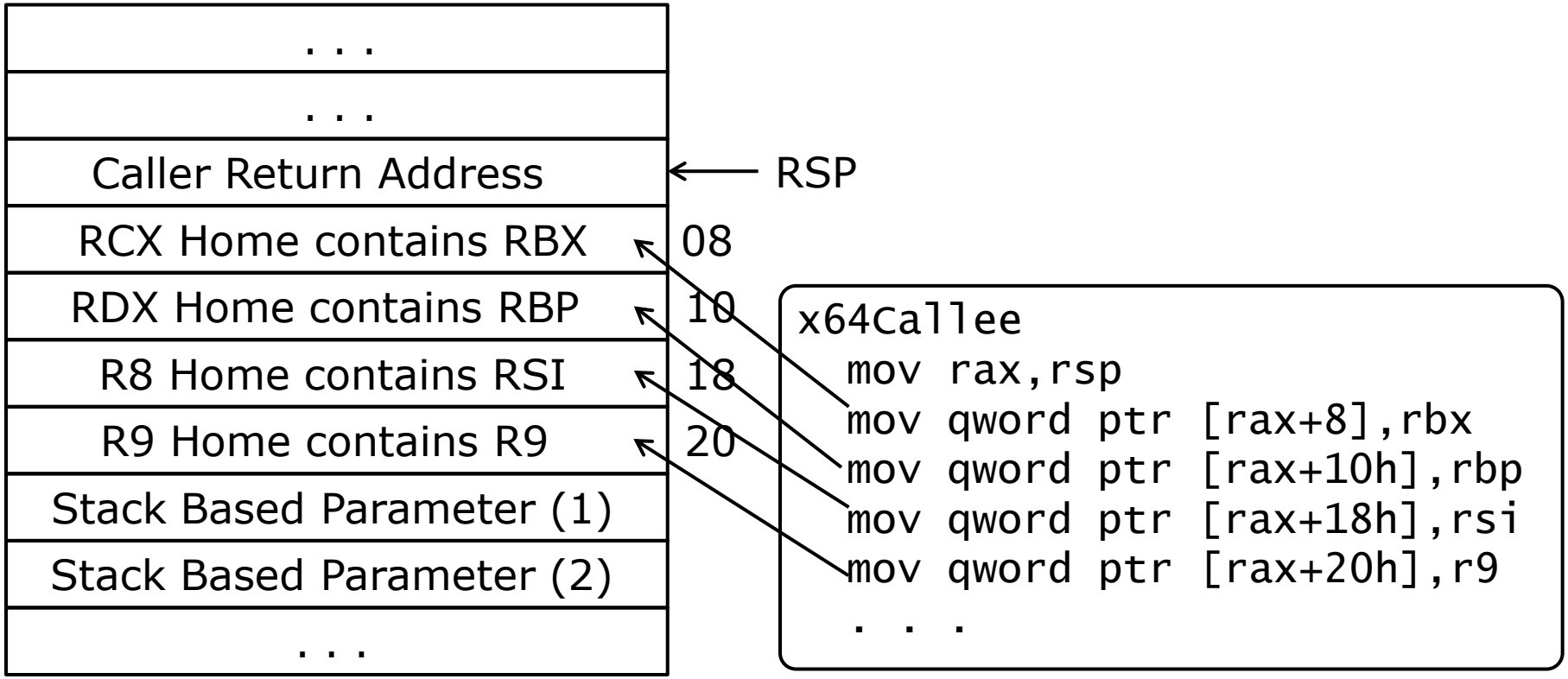
- First 4 parameters to functions are always passed in registers
- P1=rcx, P2=rdx, P3=r8, P4=r9
- 5th parameter onwards (if any) passed via the stack

```
x64Function
    . . .
    push p6
    push p5
    mov r9, p4 ; qword param
    mov r8, p3 ; qword param
    mov rdx, p2 ; qword param
    mov rcx, p1 ; qword param
    call Function1
    . . .
```

```
x64Function
    . . .
    push p6
    push p5
    mov r9b, p4 ; byte param
    mov r8w, p3 ; word param
    mov edx, p2 ; dword param
    mov rcx, p1 ; qword param
    call Function1
    . . .
```

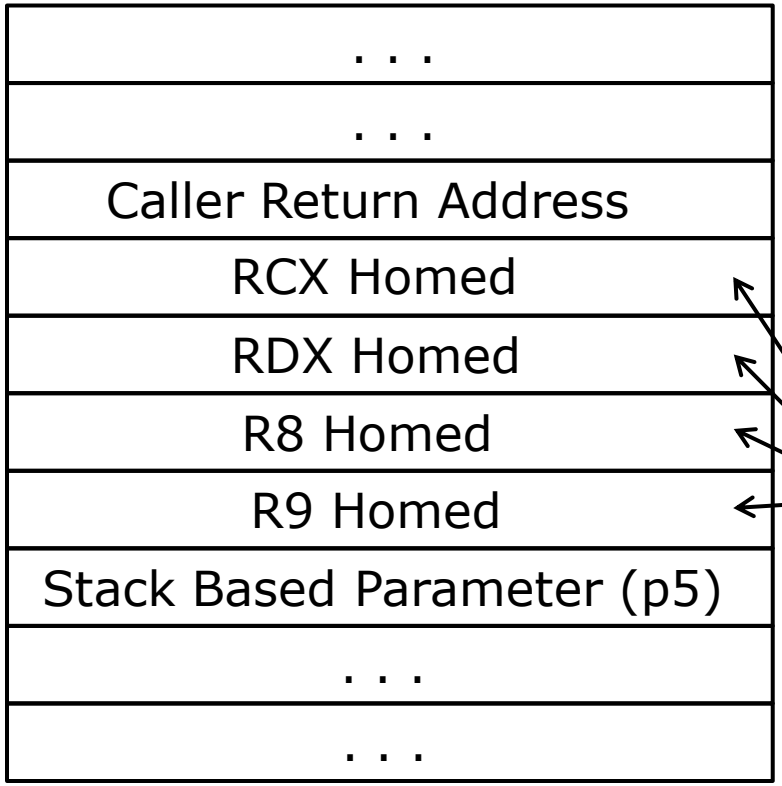
Homing Space

- Stack space allocated for register based parameters
- Minimum size of homing space is 0x20 bytes or 4 slots
  - Even if function takes less than 4 parameters
- Typically used to store NV registers



Parameter Homing

- Enabled by /homeparams compiler flag
- Disabled on free builds but enabled on checked builds
- Callee saves register-based-parameters on the stack
  - Does not affect caller

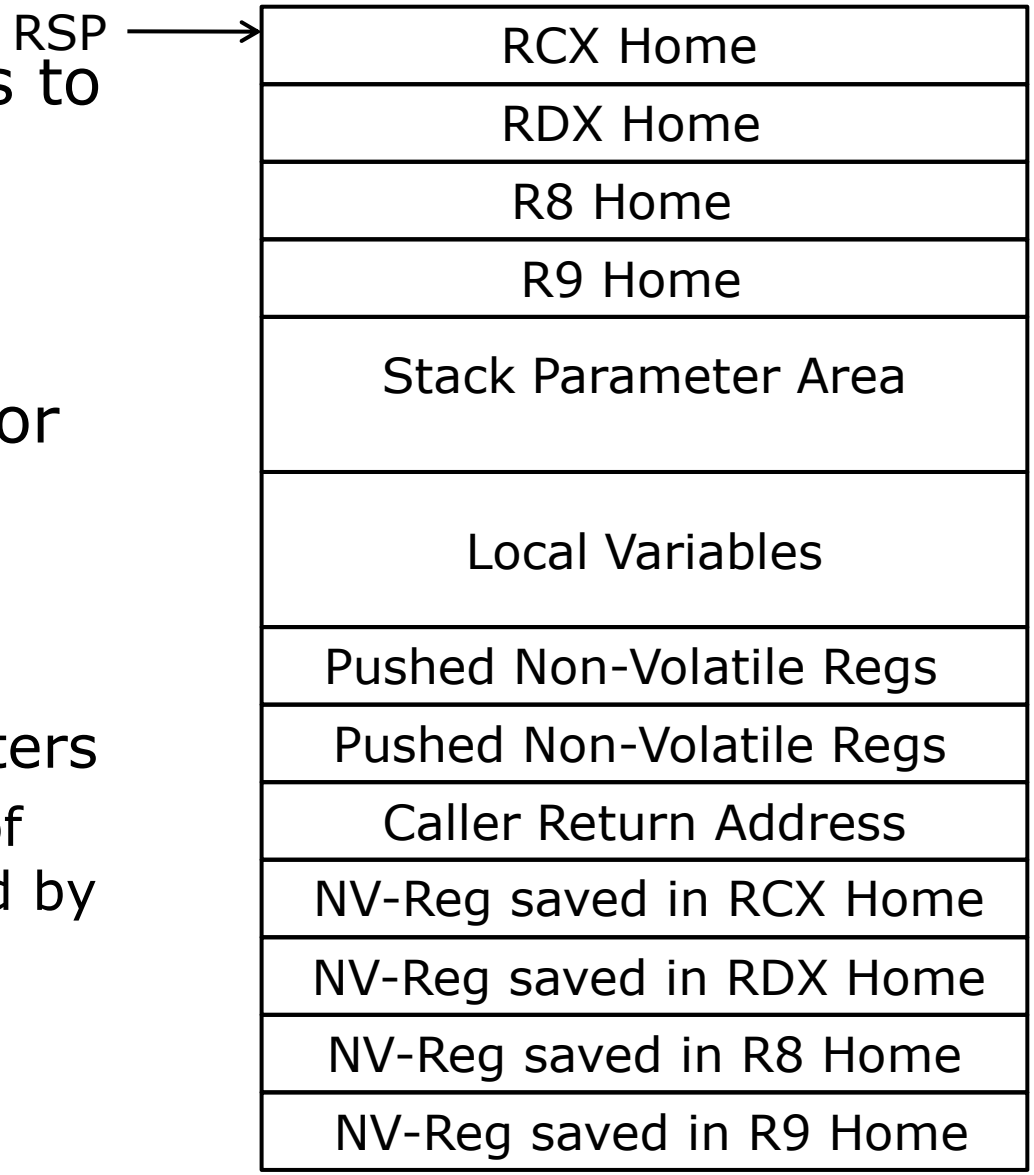



```
x64Caller
  push p5
  mov r9, p4
  mov r8, p3
  mov rdx, p2
  mov rcx, p1
  call x64Callee
```

```
x64Callee
  mov rax,rsp
  mov qword ptr [rax+20h],r9
  mov qword ptr [rax+18h],r8
  mov qword ptr [rax+10h],rdx
  mov qword ptr [rax+8],rcx
  . . .
```

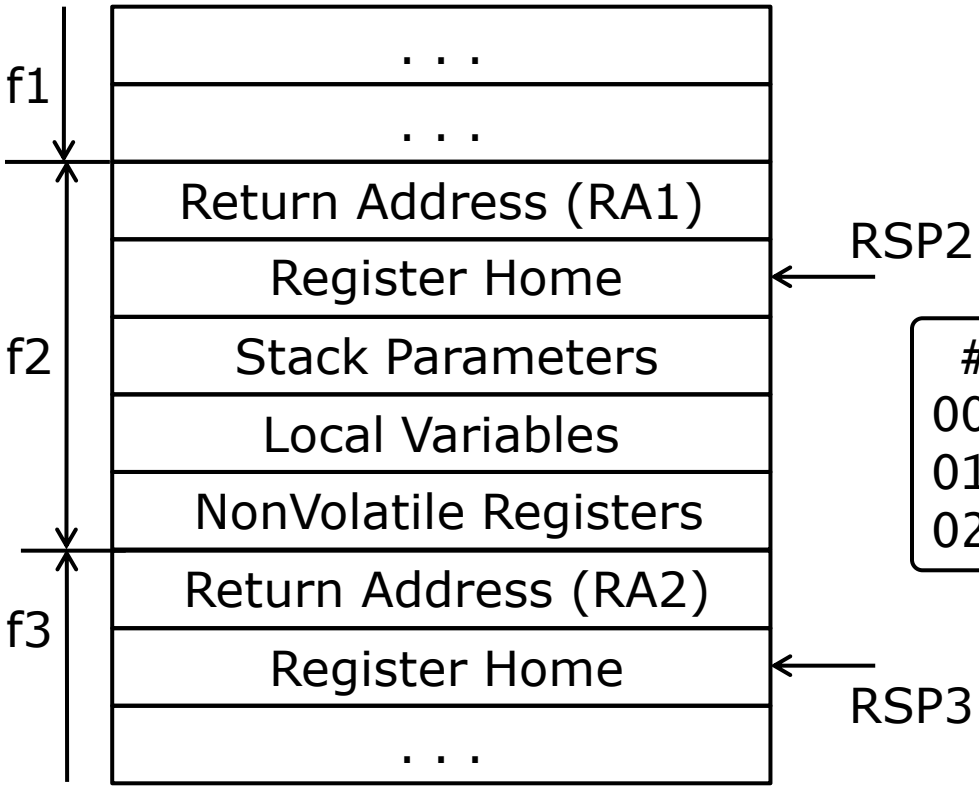

Stack Usage

- Save (move) registers to stack
- Push NV registers on stack
- Allocate stack space for
  - Locals
  - Register based parameters
  - Stack based parameters
    - Maximum number of parameters required by any function call



Child-SP

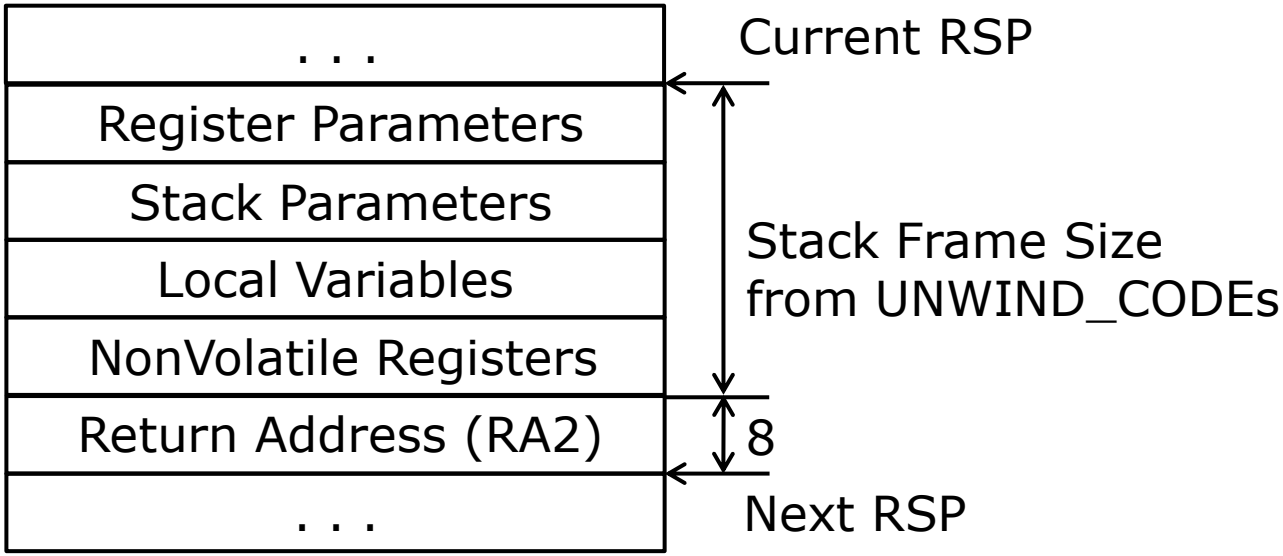
- Position of stack pointer after the function prolog
- x64 functions do not modify RSP after function prolog
- Stack based parameters and local variables are accessed relative to RSP



#	Child-SP	RetAddr	CallSite
00	RSP1	RA1	f1
01	RSP2	RA2	f2
02	RSP3	RA3	f3

Walking x64 call stack

- Unlike x86, no RBP chain on x64
- Debugger computes size of function stack frame using stack usage information in the UNWIND_CODEs
- Computes value of next Child-RSP using current RSP and size of the current stack frame



Finding Register based Parameters

- Registers get modified constantly as code executes
- Need to find where
  - The value in the register came from (parameter source)
  - The value in the register is going to (parameter destination)

For parameter destinations use current and previous frame

#	Child-SP	RetAddr	CallSite
00	RSP1	RA1	f1
01	RSP2	RA2	f2
02	RSP3	RA3	f3

To find parameters to function f2()

For parameter sources use current and next frame

Identifying parameter sources

- Works for parameters that are
  - Constant Values
  - Pointers to global data structures
  - Values from global data structures
  - Pointers to buffers on the stack
  - Values stored on the stack
- Disassemble the next frame to find the source of the values being loaded into RCX, RDX, R8 and R9

Disassembly of Previous Frame

```
x64Caller
  . . .
  push p5
  mov r9, 0x12345678
  lea r8, [module!g_Data]
  mov rdx, qword ptr [rsp+c8]
  lea rcx, [rsp+6c]
  call x64Callee
```

NV Regs as parameter sources

- Disassemble the next frame to find if the source of the values being loaded into RCX, RDX, R8 and R9 are non-volatile registers
- Disassemble the current frame to check if those NV registers are saved on the stack
- Retrieve those NV registers to find the parameters

```
x64Caller
  push p5
  mov r9, rbp
  mov r8, rbx
  mov rdx, r12
  mov rcx, rdi
  call x64Callee
```

Disassembly of Next Frame

```
x64Callee
  mov qword ptr [rax+20h],rsi
  mov qword ptr [rax+18h],rdi
  mov qword ptr [rax+10h],rbp
  mov qword ptr [rax+8],rbx
  push r12
  . . .
```

Disassembly of Current Frame

Identifying parameter destinations

- Disassemble the function in the current frame to check if the values in RCX, RDX, R8 and R9 are being saved on the stack
- Retrieve the values from the stack using the Child-ESP value for the current frame

Disassembly of Current Frame

```
x64Callee
  . . .
  mov qword ptr [rsp+3c], r9
  mov qword ptr [rsp+38], r8
  mov qword ptr [rsp+34], rdx
  mov qword ptr [rsp+30], rcx
  . . .
```

NV Regs as parameter destinations

- Disassemble the function in the current frame to
  - Check if the values in RCX, RDX, R8 and R9 are being saved into NV Registers
  - Check if these values saved in NV Registers are being kept intact till the function in previous frame is called
- Disassemble the previous frame to check if it saves those NV Regs
- Retrieve these NV registers to find the parameters

```
x64Caller
  mov rbp, r9
  mov rbx, r8
  mov rsi, rdx
  mov rdi, rcx
  call x64Callee
```

Disassembly of Current Frame

```
x64Callee
  mov qword ptr [rax+20h],rsi
  mov qword ptr [rax+18h],rdi
  mov qword ptr [rax+10h],rbp
  mov qword ptr [rax+8],rbx
  push r12
  . . .
```

Disassembly of Previous Frame

Portability Tips

- When displaying pointers use %p
  - Used in DbgPrint(), vsprintf(), RtlCchPrintf() etc
  - %x truncates off the upper 32 bits on x64
  - PreFast checks for this
- Handles are not 16 bit numbers on either x86 or x64
  - They are pointer sized
  - So treat them as such
- Use a polymorphic type like SIZE_T to store lengths
- Don’t assume most significant 20 bits of VAs are not used

Debugger commands

- When you need pointer size don’t hardcode 4 and 8
  - Use pseudo register ‘@$ptrsize’
- When you need to display a pointer don’t use ‘dd’ or ‘dq’
  - Use the polymorphic command ‘dp’
- Save some screen real estate during x64 debugging
  - Get rid of the opcodes using ‘.asm no_code_bytes’
  - Don’t display function parameters, they are invalid anyways
  - Use ‘kn’ instead of ‘kvn’ or ‘kPn’

Questions

- Ask them now …
- Email them later to msges2010@codemachine.com
- Coming Soon… at http://www.codemachine.com
  - In-Depth Technical Articles
  - Debugging & RE Tools
  - Debugger Extensions

Global Developer Support